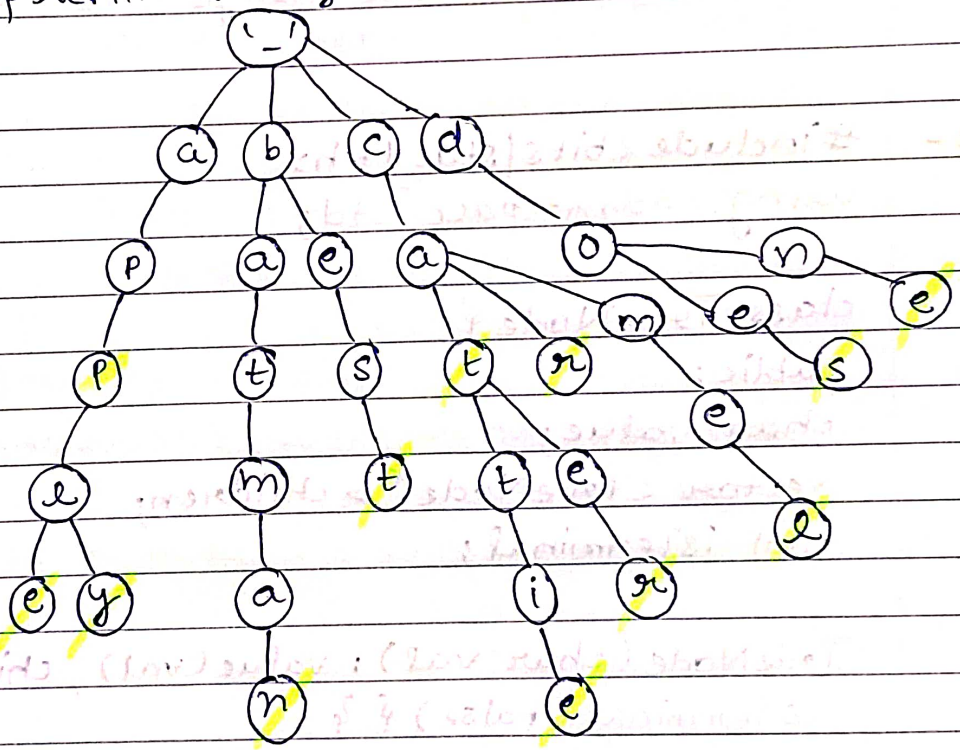
03/01/2024
Wednesday

* **Find suggestions -**

Given a trie with so many words inserted in it and a prefix string.
Return all the words in trie which comprises that prefix string.

Let the trie comprises-
app, apple, apply, batman, best, cat, car, cattie, camel, cater, does, done
and prefix string be "ca".



The strings in trie that contains prefix string "ca" is-

cat, car, cattie, camel, cater
So, this would be the answer.

Approach -

(1) Traverse till the last character of the prefix string.

(2) Find all solutions below this last character.

For instance, prefix = "cat". So, hum trie me prefix ke last character tak traverse krenge i.e. (t) tak jayenge and below that jitne bhi solutions bn rahe honge, vo answer hai.

Code -

```
#include<bits/stdc++.h>
using namespace std;

class TrieNode{
public:
char value;
vector<TrieNode*> children;
bool isTerminal;

TrieNode(char val): value(val), children(26,NULL),
 isTerminal(false){}
};
```

```
void insertWord (TrieNode *root, string word){
    //base case
    if (word.length()==0){
        root->isTerminal=true;
        return;
    }

    char ch= word[0];
    int index= ch-'a';
    TrieNode * child= NULL;

    if (root->children[index] != NULL) {
        child= root->children[index];
    }
    else{
        child = new TrieNode (ch);
        root->children[index] = child;
    }
    //recursive call
    insertWord (child, word.substr(1));
}
```

```
// ye function given prefix se start hone wali
// saari strings output krta hai

void storeStrings(TrieNode* root, string &prefix,
                  vector<string>&ans) {
    // base case
    if(root->isTerminal==true){
        ans.push_back(prefix);
    // iske baad yha se return nhi krna hai
    // because ho skta hai ki ek string ka
    // terminal mil jaye lekin aage or strings
    // exist kr rhi ho, that's why
    }

    // ye function khud khtm ho jayega jb
    // saare children process ho jayenge

    // for every trienode, saare children explore
    // kro, jo child non-null h use process kro

    for(char ch='a'; ch<='z'; ch++){
        int index = ch-'a';
        TrieNode* next = root->children[index];
        if(next){
            // 1 character store krva liya
            prefix.push_back(ch);
            // baki recursion krva dega
            storeStrings(next, prefix, ans);
            // backtrack
            prefix.pop_back();
        }
    }
}
```

```
//is function se hum given prefix ke last
// character pr pahoch jayenge

void findSuggestions(TrieNode* root, string &prefix,
                     int i, vector<string> &ans){

    //base case
    if(i == prefix.length()){
        TrieNode* lastChar = root;
        storeStrings(lastChar, prefix, ans);
        return;
    }

    char ch= prefix[i++];
    int index= ch-'a';
    TrieNode* child= NULL;
    if(root -> children[index] != NULL){
        child= root -> children[index];
    }
    else {
        return;
    }

    //recursive call
    findSuggestions(child, prefix, i, ans);
}
```

```
int main(){
    TrieNode* root = new TrieNode('_');

    insertWord(root, "apple");
    insertWord(root, "apply");
    insertWord(root, "batman");
    insertWord(root, "cat");
    insertWord(root, "car");
    insertWord(root, "cater");
    insertWord(root, "cattie");
    insertWord(root, "camel");

    string prefix = "cat";
    int i = 0;
    vector<string> ans;

    findSuggestions(root, prefix, i, ans);

    cout << "ans string:" << endl;
    for(auto i: ans){
        cout << i << endl;
    }
}
```

O/P -

```
ans string:
cat
cater
cattie
```

* **Get suggestions -**

Given is trie and prefix string. let the prefix string be - "batm".
Hume trie me "b" se start hone wale, "ba" se start hone wali, "bat" se start hone wali, "batm" se start hone wali saari strings ko ek vector of vector me return krna hai.

Code-

```
vector<vector<string>> getSuggestions(TrieNode* root,
                                      string prefix) {
  vector<vector<string>> output;
  TrieNode* prev = root;
  string currPrefix = "";

  for (int i=0; i< prefix.length(); i++) {
      char lastChar = prefix[i];
      int index = lastChar - 'a';
      TrieNode* curr = prev->children[index];

      if(curr){
        currPrefix.push_back(lastChar);
        vector<string> ans;
        storeStrings(curr, currPrefix, ans);
        output.push_back(ans);
        prev = curr;
      }
      else {
        break;
      } }
  return output;
}
```

For instance, let trie contains -

apple, apply, bat, batter, batman, best, back, cat, cater, do, done

and if prefix string be "batm".

The answer would be -

"b" -> bat, batter, batman, best, back

"ba" -> bat, batter, batman, back

"bat" -> bat, batter, batman

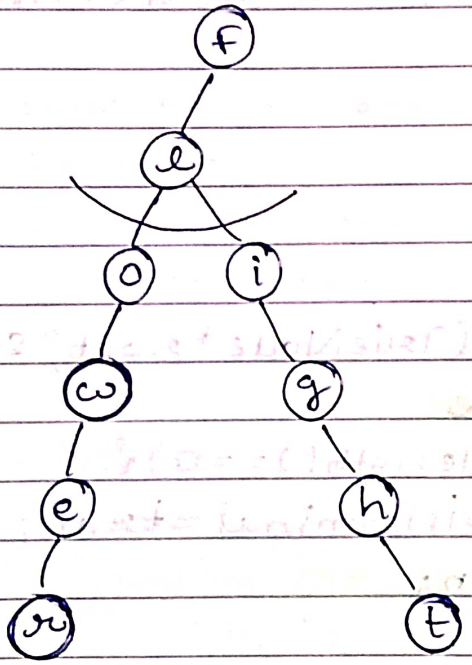
"batm" -> batman

* **Longest Common prefix -**

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

I/P - strs = ["flower", "flow", "flight"]

O/P - "fl"



Approach -

We can see here is,

- if TrieNode has single child, then ans string me store kr lo. If not, then don't store and return.

Code -

```
class TrieNode{
public:
char value;
vector<TrieNode*> children;
// child count maintain krlenge for every
// triehode
int childCount;
bool isTerminal;

TrieNode(char val): value(val), children(26,0),
  childCount(0), isTerminal(false) {}
};

class Solution{
public:

void insertWord(TrieNode*root, string word){
    //base case
    if(word.length()==0){
        root->isTerminal=true;
        return;
    }

    char ch= word[0];
    int index= ch-'a';
    TrieNode * child= NULL;

    if(root->children[index]!=NULL){
        child= root->children[index];
    }
```

```
else{
    child = new TrieNode (ch);
    root -> children[index] = child;
    // child count increase krdo
     root -> childCount ++;
    }

  insertWord ( child, word. substr(1));
}

// this function will find lowest common prefix

void  findLCP (TrieNode* root, string &ans){
      // base case
      if (root ->isTerminal ){
            return;
       }

      TrieNode * child;
      if (root ->childCount == 1){
          // child tak jao
          for (int i = 0; i<26; i++){
              if (root -> children[i]){
                    child = root -> children[i];
               }
          }
          // ans me store krao
          ans. push_back (child -> value);
      }
     else  return;

    findLCP ( child, ans);
}
```

```
string longestCommonPrefix(vector<string>& strs){

        TrieNode* root = new TrieNode('-');

        for (auto str : strs){
                insertWord(root, str);
        }

        string ans = "";
        findLCP( root, ans);
        return ans;
    }
};
```